Research on Measurement and Estimation Method for Designed Size of Aviation Equipment Software

LIU Shuang[1, a *], GUO Ji Lian[2,b] and SHEN An Wei[3,c]

[1]Graduate school, Air Force Engineering University, Xi′ an 710038，China

[2] Aeronautical Engineering College, Air Force Engineering University, Xi'an Shanxi 710038, China

[3]Refit Department of Aviation Personnel, Air Force Harbin Flight Academy, China

[a]LIU_Shuang026@163.com, [b]guojilian@163.com, [c]zjsaw@foxmail.com

Keywords: Case Point; Software Cost; Software Size; Echo State Networks

Abstract. How to estimate the aviation equipment software size accurately is the precondition of carry out the estimating software R&D cost. This paper analysis the common case point method to estimate the software size. And then the paper point out three shortcomings of the method. That is the method can't use the single point to estimate the software size. The method is discontinuity in dividing the complexity degree of use cases. The technical complexity factor, environmental complexity factor and the software size have not necessity connection. This paper put up with an improved case point method to estimate the software size. The new method distinguishes every single case clearly, and calculates every case as separate when calculating the software. The new method adjusts the use cases' weight of complexity degree. That is use continuous function curve instead of discontinuity in dividing the complexity degree of use cases. The method no longer consider technical complexity factor, environmental complexity factor. The method also considers the non-functional requirements and project risk factors when evaluating the workload and costs. The case analysis shows that the improved method can estimate the software size more accurate.

Introduction

During estimation of expenses for aviation equipment software, the measurement of software size is always the key and difficult problem drawing high attention. Software size must be measured accurately, so software expenses can be estimated accurately [1-3]. Reference [4] illustrates the close relations between software size measurement and expense estimation based on two practical cases. Professor Boehm also pointed out: “software size is the most important input of COCOMOII model”. Hence, both the academic circle and the industrial circle focus on finding a scientific, rational and effective size measurement method for accurate measurement of software size.

Zhu Anjiang [5] researched the earlier stage of software life cycle to estimate software size, and proposed an earlier function point method based on the standard function point method. With “Research and Practice of Function Point Analysis Method in Software Size Estimation” as the theme, Tu Jishan [6] researched computation rules of the function point analysis method and developed a function point computation tool supporting the implementation specifications on this basis. Above research achievements were obtained in software size estimation of general software.

Based on research of aviation equipment software, the paper analyzes common methods for software size measurement and proposes an improved use case point method aiming at defects of the traditional use case point method. The improved use case point method is changed a lot based on the traditional use case point method, so its accuracy could not be verified through direct comparison with the traditional use case method. Hence, an echo state network is introduced and relations between size and workload are established to verify accuracy of the improved use case point method [7-10].

Fundamentals of Use Case Point Method

At present, the industrial circle and the academic circle have proposed a lot of methods for size measurement. Methods concerning code line, function point and relevant extension modes, predicted object point, use case point and so forth are commonly used, wherein the use case point method is applied most widely [11].

With the technological development, object-oriented design technologies are applied to modern software design more and more extensively. The unified modeling language (UML) is used more widely for research and development. Hence, the UML-based size measurement method emerges. The Use Case Point is the most representative method[12].

Gustav Karner proposed the Use Case Point Method in 1993. Basic ideas are as follows: use cases and their actuators are recognized according to functional demands and definitions of software projects, and complexities are judged according to rules; then, use case points are computed according to the complexities in a classified manner; use case points are converted into research and development workloads according to specific conversion relations. Use steps of the use case point method are as follows:

1) Total unadjusted actuator weight (UAW) is computed. According to interaction complexities between the system and actuators, the actuators are classified into “complex”, “average” and “simple”, wherein corresponding weights are 3, 2 and 1, respectively.

$$UAW = \sum_{i=1}^{n} N \times W \tag{1}$$

2) Unadjusted use case weight (*UUCW*) is computed. It is determined according to the number of transactions of use cases described in the scene. Transaction is defined as an execution course among a series of tasks. Completion of a task depends on completion of all the transactions. Classification is conducted based on the quantity of transactions in the use case.

$$UUCW = \sum_{i=1}^{n} N \times W \tag{2}$$

Where: N is the quantity of use cases of the corresponding type; W is the corresponding weight; n is the type of use case. If the quantity of transactions is smaller than or equal to 3, we can define the use case type to be simple, with weight of 5; if the quantity of transactions is 4-7, we define the use case type to be average, with weight of 10; if the quantity of transactions is equal to or larger than 7, we deem the use case type to be complex, with weight of 15.

3) Unadjusted use case point (*UUCP*) is computed. *UUCW* and *UAW* are added together, so the unadjusted use case point *UUCP* can be obtained.

$$UUCP = UAW + UUCW \tag{3}$$

4) Technological complexity factor (*TCF*) and environmental complexity factor (*ECF*) are computed. The technological complexity factors embody influences of non-functional demands of projects on the projects, including 13 factors.

$$\begin{cases} TCF = 0.6 + \left(0.01 \times TFactor\right) \\ TFactor = \sum_{i=1}^{13} W_i \times R_i \end{cases} \tag{4}$$

Where: W_i is the weight and R_i is the relevancy. They have the same significance in computation of environmental complexity.

$$\begin{cases} ECF = 1.4 + \left(-0.03 \times EFactor\right) \\ EFactor = \sum_{i=1}^{8} W_i \times R_i \end{cases} \tag{5}$$

5) Use case point (*UCP*) and workload are computed. Use case points can be obtained according to influences of *TCF* and *ECF* on the unadjusted use case point.

$$UCP = UUCP \times TCF \times EF \quad (6)$$

While proposing the method at the very beginning, Karner suggested that the workload of each use case point was 20 man-hours. Based on Karner’s research, Ribu deemed the workload to be 15-30 man-hours. Conversion between scale and workload is as follows:

Workload (man-hour)=quantity of use case points × workload (man-hour) of each use case point (7)

Software size measurement is conducted in modern software projects which utilize object-oriented design methods more and more extensively. Research and application of the use case point method will become popular. Especially at the earlier stage of software project research and development, the software size can be measured rapidly and accurately through determination of functional demands and definitions of the software system. A lot of existing research references concerning the use case point method show that the use case point method is valid. Application of the use case point method in software size measurement has the following advantages: (1) computation is simple and convenient, while it does not cost a lot of time and workloads. (2) It does not rely on programming language and research and development technologies, and estimation can be conducted independently. (3) It can be understood easily, and the system scope can be determined rapidly according to the demand definition. (4) With the technology of structurizing, classified arrangement and analysis can be conducted more effectively. (5) The use case point method depends on system demands, while recognition is simple and visualized. (6) The use case point method can be applied to other measurement models and can measure the software scale as the basis. In addition, the use case point method is centered on users and based on use rather than be design-oriented or system-oriented, so it is more stable and robust than other methods.

In object-oriented software design and R&D, the use case modeling has been widely used by R&D staffs and is applied to capture and presentation of functional demands of systems at the earlier stage of software R&D. Use case is an obvious and simple idea. Use case defines frame and functional demands of projects at the earlier project stage. Hence, during measurement of software size, the use case point method can be used to estimate size of R&D software and estimate software expenses.

Through profound research and analysis of the traditional use case point method, we could find a lot of advantages of the use case point method as well as some deficiencies. Detailed analysis will be conducted as follows.

Failure to Measure Single Use Case. In management of software R&D projects, distribution work is often conducted based on use cases. Hence, size and corresponding workload of each use case must be known, and project plans must be formulated on the basis. However, the traditional use case point method cannot estimate single use case, retarding project management and enlarging errors of software size estimation. On the other side, during computation of unadjusted actuator weights, the traditional use case point method takes software project or functional system as the units in general. In other words, all the use cases contained by the whole project or system are measured. It is the same with measurement of actuators. Therefore, actuators contained by each use case in the project or system cannot be defined and quantified.

Discontinuity in Grading of Use Case Complexity. During computation of unadjusted use case weights, the complexity of use cases depends on the quantity of transactions contained in use cases. However, the traditional use case point method is discontinuous in complexity grading, so the determined complexity weights are not accurate. Three use cases are used as the case for illustration, as shown in Table 1.

Table 1 Determination of Use Case Complexity Wight with Traditional Use Case Point Method

Project	Use Case 1	Use Case 2	Use Case 3
Quantity of transactions	3	4	6
Complexity	Low	Average	Average
Weight	5	10	10

According to several typical situations given in the table, we can find: (1) According to the traditional determination rules of use case complexity weights, use case 2 and use case 3 are classified to the "average" complexity, and the same weight 10 is distributed. However, use case 3 has 2 more transactions than use case 2. Obviously, the use case 3 is more complex. (2) Use case 1 containing 3 transactions is classified to the low complexity, and the corresponding weight 5 is distributed. However, if one transaction is added for use case 1 (use case 1 is transformed to use case 2), the use case 1 will be classified to the "average" complexity according to the complexity matrix.

According to the analysis results, we can find that the traditional use case point method is disadvantageous in division discontinuity during determination of complexity weights of use cases, while the use has certain randomness. If these situations appear in the same software project, the measurement results will not satisfy actual situations. Especially under huge quantity of use cases, errors will be huge.

No Inevitable Relations between Technological Complexity Factors, Environmental Complexity Factors and Software Size. Technological complexity factors and environmental complexity factors are mainly used to reflect nonfunctional demands of software projects as well as influences of project risks on software research and development, but they have no inevitable relations with inherent size of software. Software size depends on functional demands defined by software projects or tasks to be completed. The software size will be larger when there are more functions which are more complex or tasks to be completed are more difficult. On the contrary, if the functions to be achieved are fewer or less difficult or the tasks to be completed are easier, the software size will be smaller.

Technological complexity factors and environmental complexity factors will not influence functions to be achieved or tasks to be completed by a software system. For example, distributed system, easiness in installation, easiness in use or other technological complexity factors are unrelated with functions to be achieved by software. Hence, they do not have direct influences on measurement of software size. For another example, development experience of application program, ability of leading analysis personnel, difficulty of programming language and other environmental complexity factors are also unrelated with functional demands of software. Hence, they do not have direct influences on measurement of software size.

Therefore, technological complexity factors and environmental complexity factors do not have inevitable associations with software size. They shall not be considered during size estimation. Their influences shall be considered during estimation of workloads and expenses.

In conclusion, the traditional use case point method is deficient in measurement of software size. It will give rise to large errors in specific use. Especially when the software system is large and complex, errors will be huge. Hence, in order to measure the software size more effectively, it is necessary to improve the traditional method so as to obtain more accurate size measurement results and lay a good foundation for estimation of software workloads and expenses.

Research on Improved Use Case Point Method

Code line method and the method of functional point and its extension have some problems in traditional software size measurement, but they still achieve good effects. However, in modern software engineering oriented to object design, it could hardly be effectively applicable and could not embody the feature of software research and development of object-oriented design. Through prediction of object points, the software size of object-oriented design can be measured effectively, but its application is complicated, while confirmation and investigation at the later R&D stage could hardly be achieved. The use case point method has unique advantages in solution of methods concerning modern software size measurement, but it still have some problems. This section conducts research based on defects of the traditional use case method.

Aiming at defects of the use case point method analyzed and discussed in the previous section, improvements are made to the traditional use case point method here. Based on the traditional use case point method, each use case is distinguished explicitly, and each use case is computed separately during size computation. The computation mode of use case complexity weights is adjusted. Continuous function curve takes the place of discontinuous grading in the traditional use case point method. During computation of software size, technological complexity factors and environmental complexity factors are not considered any more. Instead, nonfunctional demands and project risk factors reflected by technological complexity factors and environmental complexity factors are considered during estimation of workloads and expenses.

The improved use case point method distinguishes single use cases definitely and computes size of each use case independently. The improved use case point method deletes grading rules of use case complexity weights, and conducts computation directly according to the quantity of transactions. In this way, the problem of large errors in measurement of software size with the traditional use case method can be solved, while each use case conversion point can be computed explicitly.

Use steps of the improved use case point method are as follows:

1. Actuator weight (*AW*) of each use case is computed. Computation method and steps of *AW* are kept the same with computation of *UAW* in the traditional use case method. According to complexity of interactions between actuator and system, it is classified into simple, average and complex. A weight factor is assigned for each type, as shown in Table 2.

Table 2 Distribution Table of Actuator Complexity Weight Factors

Complexity	Description	Weight
Simple	External system with a defined application program interface (API)	1
Average	Interaction system in need of a communication protocol (such as TCP/IP); or person conducting interactions with system by terminal	2
Complex	Person interacting with application programs by graphic interface or W’EB page	3

Weight of actuator (*AW*) for each use case is computed by the following formula:

$$AW = 1\times simple + 2\times average + 3\times complex \tag{8}$$

Where: $simple$ is the quantity of simple actuators; $average$ is the quantity of average actuators; $complex$ is the quantity of complex actuators.

2. Use case weight (*CW*) of each use case is computed. Use case weight shall be represented by the quantity of transactions contained in each use case. If computation is conducted after classification, as stipulated in the traditional use case method, errors will be large and relations between the quantity of use case transactions and software size cannot be reflected accurately. Hence, analysis results of a lot of historical data indicate that computation of use case weights based on direct use of transaction quantity has better effects. Meanwhile, we can find that the quantity of transactions not only influences use case weight, but also has significance influences on size of use case point. Hence, during computation of use case weights and use case points, computation formulas of exponential form or power function form are used. Through analysis of a lot of data concerning use case weights and transaction quantity, we can fit the following formula:

$$CW = 15\times(1-0.8^{T}) \tag{9}$$

3. Use case point (*UCP*) of each use case is computed. Size of use case point depends on actuator weight and use case weight. If a use case contains actuators and transactions with larger quantity and higher complexity, namely actuator weight and use case weight are larger, the size of use case point will be larger. On the contrary, if a use case contains fewer or less difficult actuators and transactions, namely actuator weight and use case weight are smaller, the size of use case point will be smaller. If actuators contained in each use case are unchanged, the sue case size will be larger if there are more transactions; otherwise the situation will be contrary. If a single use case has certain quantity of transactions, the use case size will be larger when there are more actuators with higher complexity. Through analysis of a lot of data, we fit the computation formula (10) of single use case point, as follows.

$$UCP = AW^{0.5} \times CW^{1.09} \tag{10}$$

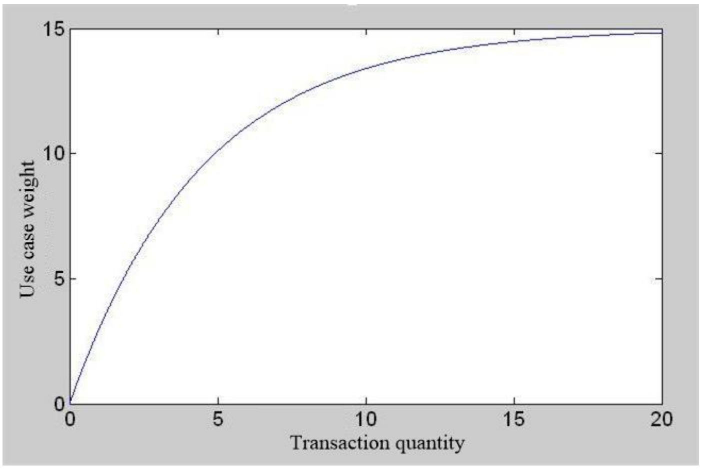

Fig.1 Relation Curve of Use Case Weight and Transaction Quantity

4. Total use case point size of software is computed. All the single use case points in the whole software system are added together, so the total use case point (*TUCP*) is obtained.

$$TUCP = \sum_{i=1}^{n} UCP_i \tag{11}$$

According to formulas (8), (9), (10) and (11), the size of software to be estimated is obtained. Based on the obtained software size, software expenses can be estimated on this basis.

Result Verification Based on Echo State Network

After improvement of the traditional use case point method, it is not verified. The use case point method improved in the paper is quite different from the traditional use case point method. Use cases in software are computed separately. Methods for computing use case complexity weights and single use case size are changed. In computation of use case size, influences of nonfunctional demands and project risks reflected by technological and environmental complexity factors on the size are not considered. In view of above reasons, the improved use case point method cannot be compared directly with the traditional use case point method. It is only feasible to compare software workloads estimated finally by the method with software workloads estimated by other methods and actual workloads. On this basis, accuracy of the improved use case point method can be evaluated.

The section selects the novel algorithm of echo state network. A model for conversion from the software size measured by use case points to workload is established. Technological complexity factors and environmental complexity factors which are not considered in size measurement are added through adjustment and taken as input of the conversion model. Other important influential factors are added based on research and analysis. Workloads are computed by the model.

No fixed formula is set for conversion from software size to workloads. Summarization and analysis are carried out mainly through continuous collection and accumulation of historical data, as well as application of methods such as statistical regression. Accurate and reliable estimation formulas can be obtained. The software size-workload conversion model can be formed gradually.

Analysis of Traditional Conversion Model. At present, most software size-workload conversion models are established on the basis of an empirical model (Cover1988), such as the traditional use case point method. The general math expression of these conversion models is as follows:

$$Effort = a \times Size + b \tag{12}$$

Where: *Effort* is the workload; *Size* is the software size; *a* and *b* are parameters, which are obtained through regression analysis of historical data. Two typical models for linear conversion from software size to workloads are as follows:

(1) Albrecht/Gaffnef Model

$$Effort = -13.39 + 0.054 Size \tag{13}$$

(2) Maston/Barnett/Mellichamp Model

$$Effort = 585.7 + 5.12 Size \tag{14}$$

However, a lot of research results indicate that nonlinear relations exist between size and workload. For example, Boehm added a scale factor into COCOMO series models to embody nonlinear relations between software size and workload; in addition, the estimation model of Putman also demonstrates the nonlinear relation between software size and workload.

Echo State Network (ESN) is a novel neural network. In comparison with the traditional neural network, it is characterized in that a reserve pool is composed of nerve cells which are connected in a random and sparse manner. The reserve pool method determines a brand new signal processing mechanism, so the nonlinear problem can be solved easily by linear methods.

Based on above analysis, ESN is used to establish a software scale-work load conversion model.

Basic steps for prediction of workload with the software size-workload conversion model based on ESN are as follows:

Step 1: Parameters such as reserve pool size *N*, reserve pool input unit scale *IS* and reserve pool sparsity *SD* are selected. According to these parameters, connecting weight matrixes $\mathbf{W}_{in}$, $\mathbf{W}$ and $\mathbf{W}_{back}$ in ESN are generated randomly. As for the connecting weight matrix $\mathbf{W}$ generated randomly, the spectral radius *SR* must be judged, namely whether λ_{max} is smaller than 1 is judged. If $\lambda_{max} > 1$, the connecting weight matrix must be re-generated randomly.

Step 2: Project data with a certain quantity is taken as training samples. Abnormal data in project data is recognized and processed.

Step 3: Arranged project data is input into the model. Echo response signals of the reserve pool are excited. According to the connecting weight matrix $\mathbf{W}$ between nerve cells inside the reserve pool as well as the target signal $\boldsymbol{y}$, the connecting weight matrix $\mathbf{W}_{out}$ can be solved.

Step 4: A part of project data other than the training sample is taken as the testing sample and used to verify accuracy of model estimation.

Step 5: Size, organizational productivity and functional complexity of the software project to be estimated are input. The estimated workload can be obtained by the trained network model.

According to the collected original data of 120 projects, the improved use case point method is used to measure size of each software size. TRUE S software is used to obtain organizational productivity and functional complexity of each software project. 110 projects are drawn randomly and taken as the training samples, the rest 10 projects are taken as testing samples. The original data is secret-related, so the original data is adjusted before simulation with the original data. However, research of the method and reliability of the results are not influenced.

Dimension of the input layer is defined to be $K = 3$. The quantity of nerve cells in the reserve pool is $N = 200$. Sparsity of connection of nerve cells inside the reserve pool is $SD = 3\%$. The input unit scale of reserve pool is $IS = 0.8$. The dimension of input layer is $L = 1$. The connecting weight matrixes $\mathbf{W}_{in}$, $\mathbf{W}$ and $\mathbf{W}_{back}$ are initialized randomly. The spectral radius $\mathbf{W}$ of satisfies $\lambda_{max} < 1$.

110 training samples are input in succession. According to state variables of reserve pool nerve cells in the updated model, during study and training, the $\mathbf{W}_{in}$, $\mathbf{W}$ and $\mathbf{W}_{back}$ are kept unchanged, and the connecting weight matrixes are output and computed after training.

In this way, 10 testing samples can be input into the model, and accuracy of the model and validity of the method can be detected. Workloads of 10 testing samples are estimated by different methods (for easy comparison, workloads computed with the ESN have been rounded). Results are shown in the following table.

It is shown in Table 4 that estimation results obtained by the ESN-based improved use case point method are obvious smaller than estimation errors of the traditional use case point method, indicating that the software size-workload conversion model based on ESN has very high accuracy. These results also verify high effectiveness and accuracy of the improved use case point method.

Table 3 Estimation Results and Accuracy of Different Methods

Sample Project	Actual Workload (Man-hour)	Traditional Use Case Point Method			Improved Use Case Point Method Based on ESN		
		Size (Use Case Point)	Workload (Man-hour)	Error	Size (Use Case Point)	Workload (Man-hour)	Error
1	6852	268	5360	0.2177	154	6125	0.1061
2	7328	295	5900	0.1949	197	6674	0.0892
3	7649	422	8440	0.1034	318	7931	0.0369
4	8391	351	7020	0.1634	245	7662	0.08688
5	8964	362	7240	0.1923	255	8257	0.0789
6	9514	368	7360	0.2264	260	8596	0.0965
7	10622	694	13880	0.3067	586	11955	0.1255
8	12658	502	10040	0.2068	405	11758	0.0711
9	13294	836	16720	0.2577	718	14358	0.0800
10	13681	874	17480	0.2777	754	15024	0.0982

Summary

Based on introduction to typical software size measurement methods, the paper analyzes advantages and disadvantages of different methods and mainly describes superiority of the use case point method. Aiming at defects of the traditional use case point method, the paper improves the traditional use case point method with new ideas. The improved use case point method is changed a lot in comparison with the traditional method, so it cannot be directly compared with the traditional use case point method. The ESN-based software size-workload conversion model was established to verify effectiveness of the conversion model and accuracy of the improved use case point method. Determination of the software size with the improved use case point method lays a solid foundation for later establishment of an estimation model for software expense parameters of aviation equipment.

References

[1] Chen Weilin, Zhang Zhengmin, Geng Huixin, Wang Xiaohan. Optimized simulation for control of military product acceptance cost [J]. Computer Simulation, 2015,(9):34-38

[2] Liu Li, Zhu Xiaodong, Ye Fei, Wang Yigang. Prediction of structural level maintenance time of software system [J]. Computer Simulation, 2013,(11):238-241

[3] Fu Yafang, Liu Xiaodong, Li Yanjie. Software size estimation method based on improved FPA [J]. Computer Engineering and Application, 2011, 47(1):22-25

[4] Li Mingshu, He Mei, Yang Da et al. Method and application of software cost estimation [J]. Journal of Software, 2007, 18(4): 775-795

[5] Zhu Anjiang, Research and application of software size estimation methods at earlier stage [D] National University of Defense Technology, Changsha, 2011

[6] Tu Jishan. Research and Practice of Function Point Analysis Method in Software Size Estimation [D] East China Normal University, Shanghai, 2006

[7] Jones C. Applied Software Measurement—Assuring Productivity and Quality. New York: McGraw-Hill Inc., 1991

[8] Vogelezang F. COSMIC full function points the nest generation of functional sizing. In: Software Measurement European Forum—SMEF 2005. 2005

[9] Xie Chenggang, Cao Wenzhao. Hybrid assessment method for cost risks of software projects [J]. Computer Simulation, 2009,(4):304-307

[10] Contact way: Zhou Yang, Wu Haitao, Zhang Dongwei. Extended use case point estimation method [J]. Computer Technology and Development, 2006, 16 (12): 64-66

[11] Ribu K. Estimating Object-Oriented Software Projects with Use Cases[D]. Norway: University of Oslo, 2001

[12] Guo Heqing. Modern Software Engineering [M]. Guangzhou: Press of South China University of Technology, 2004